Fancy some Chips for your TeaStore? Modeling the control of an adaptable discrete system *

Anna Gallone
Université Marie et Louis Pasteur, CNRS UMR6174, Institut FEMTO-ST, F-25000 Besançon, France
anna.galdel@protonmail.com

Simon Bliudze
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
FirstName.LastName@inria.fr

Sophie Cerf
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
FirstName.LastName@inria.fr

Olga Kouchnarenko
Université Marie et Louis Pasteur, CNRS UMR6174, Institut FEMTO-ST, F-25000 Besançon, France
Olga.Kouchnarenko@femto-st.fr

When designing new web applications, developers must cope with different kinds of constraints relative to the resources they rely on: software, hardware, network, online micro-services, or any combination of the mentioned entities. Together, these entities form a complex system of communicating interdependent processes, physical or logical. It is very desirable that such system ensures its robustness to provide a good quality of service. In this paper we introduce Chips, a language that aims at facilitating the design of models made of various entwined components. It allows the description of applications in the form of functional blocks. Chips mixes notions from control theory and general purpose programming languages to generate robust component-based models. This document presents how to use Chips to systematically design, model and analyse a complex system project, using a variation of the Adaptable TeaStore application as running example.

1 Introduction

As websites become more and more permissive, an increasing number of technologies can be put as back-end of the services they provide, thus making web applications a breeding ground for the development of very complex systems. To ask for a restaurant near our current position means to involve a cartography API, a recommendation module, a GPS protocol and a shortest path algorithm. Each one of these components can be hosted respectively on a company distant server, the user's device, satellites and the website provider's server. To take decisions about what processes to handle remotely or how to tune algorithms efficiently can quickly become hard. Models can help make such decisions before investing resources in the implementation of a "complex system", a term that will be used hereafter to describe sets of communicating processes, each process potentially being implemented by subsystems, which may be complex as well.

*This work was supported by the ANR grant ANR-23-CE25-0004 (ADAPT). O. Kouchnarenko was supported by the EIPHI Graduate School (grant number ANR-17-EURE-0002).

Submitted to:
Workshop on Adaptable Cloud Architectures
DisCoTec 2025 - 20th International Federated Conference
on Distributed Computing Techniques

© A. Gallone et al.
This work is licensed under the
Creative Commons Attribution License.

In the context of Cyber Physical Systems (CPS) that are subject to uncertainties, the Chips language is designed with the aim to facilitate the design of CPSs' models. Chips standing for Control of Hierarchical Interconnected Programmable Systems, has a synchronous syntax and original features that permit using Control Theory to palliate those uncertainties. The language provides the tools to both represent a complex system's significant behaviors and develop additional components tuning the devices parameters at run-time. Chips compilation into BIP [5] opens the way for many additional model based operations. Indeed, BIP (for Behavior Interaction Priority) is a framework for developing interacting automata based models. Its rigorous implementation of synchronicity mechanisms makes it possible to derive these models to correct-by-construction programs generation [8], and model exploration tools are provided alongside the BIP compiler to ensure the right behavior of the designed applications.

When it comes to the adaptation of a complex system, one can distinguish two kinds of adaptation for programmable components: behavioral adaptation and structural reconfiguration [9]. The former is about modifying the data manipulated among the components for the system to take decisions, while the latter is about modifying the way components interact, by modifying the number of instances of subsystems in use or their connections. Behavioral adaptation can be achieved with the BIP framework pretty easily, while structural reconfiguration requires more work, hence the development of DR-BIP (Dynamic Reconfigurable BIP) [4]. Providing a language to automatize the generation of BIP models should allow achieving adaptation goals in a systematic way. In this context, Chips proposes to take a step back from the BIP models and to add upon them another abstraction layer taking advantage of two main concepts:

- The first one being *Control Theory* (CT for short) [17], a well known engineering domain, where each component is modeled by a function. Additional control components allow the system behavior to adapt at run-time. By assembling the functions together, CT turns a physical system into a differential equation. Such equation solutions are the different behaviors to adopt for the control components so the system realizes a task as expected. Though control theory is mainly applied to the engineering of continuous systems, it can find applications for discrete systems too [12][1].
- The second concept is *Aggregate Programming*, a programming paradigm implementing collective communication primitives that rely on field calculus [16]. The role components take in the collective behavior of the system is computed on the go by the state of variables continuously updated on each device according to their position. In such systems, leadership is assumed by the field of the values shared in the space and not by a particular device, making applications developed according to the aggregate programming principles extremely resilient to structural changes.

By including these concepts in the BIP model workflow, Chips gives the means to easily tune both the behavioral and structural adaptation of any complex system. This document shows how Cloud Computing (CC) can benefit from this approach on the Adaptable TeaStore example [2][7]. The described experiment serves as a proof of concept for the applicability of CT formalism to computer systems. Following a precise methodology, it explains how one can devise an adaptive version for any component of a complex system. Here, the emphasis is put on the adaptation of a cache component for the TeaStore website server to improve its response time to users requests.

In the rest of this paper, Section 2 details a preliminary analysis of the requirements for the Adaptable TeaStore. Section 3 presents the modeling framework by introducing Chips features and the followed methodology. This leads to the evolution of the analyzed material into a general model presented in Section 4. Then, in Section 5 simulation runs are presented, validating the behavior of the described system. Finally, after concluding in Section 7, more insight is given in Section 8 about ongoing work directions to improve this analysis-model-test pipeline relevance.

2 TeaStore requirement analysis

We start by an analysis of the requirements associated with the Adaptable TeaStore case study [2][7]. When working on the control of a system, two main signals have to be identified, the goal and the knobs signals. The goal signal is the ideal state in which the system should be. For example, this could be the response time, the overall CPU load or memory use of a system. The goal signal serves as input for the controller component introduced by CT to make the system adaptable. On the other hand, the knobs are the signals produced by such controller component. They aim at changing the system behavior toward desired goals. To identify them, more domain knowledge is required because knobs signals can sometimes overlap each other or lie in complex parts of the system. For example, if the goal of the controller is to ensure a certain response time, the knobs could correspond to the (de)activation of processor overclocking or to the modification of the available cache size.

Goals identification Acknowledging common practices of web development, the main goal of the TeaStore application should be to provide an enjoyable online shopping experience. Adaptable TeaStore case study specification [7] distinguishes mandatory modules, and optional ones. It could be assumed that the mandatory modules are in charge of the base functionalities the application offers, while the other are additional non-functional requirements to fulfill.

The modules suggested are (mandatory modules in bold font): **Web User Interface** (WebUI), **Persistence service**, **Image providing service**, Authentication functions, recommendation system.

On the functional requirements side, it should contain images of the articles the store has to sell, suggest items to buy, allow the users to navigate through the pages anonymously and store items in a cart to buy them later. On a non-functional side, one could imagine designing an application achieving these goals within a certain time to hold the user's attention, adding authentication services through other websites accounts, having a minimal electrical consumption, actively monitoring the clients behaviors to enhance the suggestion algorithm, etc.

Any of these functionalities can be submitted to thorough analysis for a CT application. Our study focuses solely on keeping the response time of the application reasonable as it is a common concern for many websites.

Knobs identification Now it has been decided what goal to follow, it is possible to identify for each component of the application what are the levers to improve timing performances of the system at execution time. Table 1 presents some of the knobs identified for the TeaStore example application, applying control to which could be of interest to reduce delays between operations. For instance, if the user internet connection is unstable, it becomes relevant to turn off image resizing to avoid sending multiple times the same picture if need be to re-send images when packets are lost.

The experiment conducted in this paper is focused on the development of the **Image provider**, and, more precisely, on how to use Chips to model a system where load balancing is achieved through the dynamic control of the local cache size. The greatest the number of elements stored in the cache, the lowest the number of requests to the database, and therefore, the less time it takes to answer a user's request.

Module	Knob name (and nature)	Description
Web UI	JS (Boolean)	Enable or disable JavaScript if unsupported operation for a browser
Image Provider	Cache size (Natural)	Adjust the number of elements the local cache can store
	Image size (Natural)	Default size of the images to store in the cache
	Image resizing (Boolean)	Authorize the resizing of images
Persistence	Request type (Categorical)	"fulldb","available","batch-queries", "top-popularity", ...
	User profile (Boolean)	Whether to retrieve additional personal data or not
Authentication	Available (Boolean)	Authorize authentication or not
	Auth kind (Categorical)	"SSOonly","TeaStoreAuth only","AnyAuth", "None", ...
Recommender	Nb params (Natural)	Number of parameters for the regression algorithm

Table 1: TeaStore Modules and their associated knob for real-time adaptation of services speed

3 Modeling framework

To properly develop a useful model for the Adaptable TeaStore application, it is important that the design process follows a formal and repeatable method. Next sections are about that precise methodology we apply to the case study, and about the Chips language involved in the design process.

3.1 Methodology for building a useful Chips model

The Chips workflow is based on the one introduced by Filieri et al. [1], section III.CT turns the components and the signals they treat into equations. Solving them allows the developer to find the best function that will push the state of the system toward a desired state. Once the solution functions are found, it is possible to design a component that will introduce their behavior into the system so the system acts as intended. As described in Section 2, our main signal to track is the response time of the server. To keep it low, the server should act at run-time on the size of the cache it uses to reduce the number of requests to the database, a time consuming operation. The next step is to devise a model representing these signals and the different functions that produce or use them: Section 4 presents the Chips description of a general model obtained. On this base, it becomes then possible to work more precisely on the controller component that will tune the chosen knob –the cache size. As Chips can be compiled to a BIP model, the object files produced can run simulations and be re-compiled for real world devices. Therefore, designing a model and a controller with Chips also means to implement them for a direct use. In this paper, Section 5 describes a simulation run to validate the behavior of the described system.

3.2 Chips language features

CC systems share many characteristics with structured distributed CPSs aimed by Chips, namely, the coordination of multiple devices over communication protocols, the repeating entry and exit of devices

in the network or the need to maintain a service when a component is temporarily unavailable. Hence, Chips can be effectively used for the design of CC systems as illustrated by Chips implementation of the Adaptable TeaStore case study, presented in Section 4.

The closest to control theory programming paradigms one can find in the state of the art is synchronous programming. Languages implementing this paradigm aim at describing flows of data instead of variables, and they transform dataflows assuming some operations can be made instantaneously. Lustre [10] and one of its derived language Heptagon [11], are respectively compilable in C and Ocaml, more recently, Eclat [15] for the description of FPGA logical circuits. Such langages provide mathematical properties that make them reliable for the design of (potentially critical) reactive systems. The description of a Chips systems makes them behave in a similar way. Chips represents each component of an application by a functionnal block with inputs and outputs, and synchronizes the execution of interconnected components so that the system modelizes some actions as simultaneous. Basic Chips constructs include different kind of functions:

- "pure", to factorize expressions that may be used elsewhere, in the components' or the system's description (lines 1 to 7 in Listing 1),
- "logical" if the function models a component of our system, it can include memory in the form of inner variables or execute sequential algorithms (lines 9 to 30 to model the different interacting modules of the application),
- or "physical" if the function models a hardware component comprising calculation capacity and allocatable memory (lines 34 and 35 to specify the way a server interfaces the virtual modules and the physical world).

In contrast to previously mentioned languages, Chips is only partially synchronous. Though logical and physical components are modeled as simultaneously executed, Chips describes the relation between variables in an imperative way, closer to general-purpose programming languages. No parallelism is admitted within the shell of a logical or physical function. Each component's inner variable is modifiable with a C-like syntax. Still, according to synchronous programming principles, the output parameters of a component are only accessed by other components once the whole *Then* section is completely executed, thus hiding the sequential logic from the whole system point of view. Such constraint ensures the atomicity of the components. Since everything that can be parallelized is set in a different component, the work of splitting apart the different functions among the modeled physical devices is simplified.

```
1  // extract last bit value for validation
2  pure user_action_is_valid(int user_action_data)
3    -> ( (user_action_data & 0x1) == 0x1 )
4
5  // all other bits are the qty of images to fetch
6  pure user_action_nb_imgs(int user_action_data)
7    -> (user_action_data >> 1)
8
9  logical interpretation_module(int user_action) init {
10   int nbr_imgs = 0;
11 } then {
12   if(!user_action_is_valid(user_action)){
13     nbr_imgs = 0;
14   } else {
15     nbr_imgs = user_action_nb_images(user_action);
16   }
17 } -> (nbr_imgs)
18
19 logical image_provider(int nb_imgs_to_provide) init {
20   int [] cache = empty_set;
21   time_since_last_req = 0;
22 } then {
23   for i in n_up_to(nb_imgs_to_provide) {
24     img_to_fetch = rnd_img_id();
25     if( !find(img_to_fetch,cache) ){
26       // simulating db request
27     }
28     lru_update(cache, img_to_fetch);
29   }
30 } -> (1)
31
32 import "server.json" as server;
33
34 physical server(int server_resources, int user_action)
35   -> (server_resources, user_action)
36
37 /* ... other definitions ... */
```

Listing 1: Code sample of Chips simplified TeaStore model components

Additionally to a Chips program, hardware description files should be provided for each physical component (see `import` command in Listing 1, line 32). Such files would give information about the components capabilities: number of processors, memory size, clock rate, actuators and sensors, etc. A Chips compiler could then determine how to make physical components interact at simulation time according to which device each function/component is associated to, or verify that a program fits a certain hardware. These constraints are specified by the way a system is assembled in the `SYSTEM` section of a model with the `link` operator (lines 7 and 8 of Listing 2).

```
1  SYSTEM {
2    user user_instance;
3
4    server server_instance;
5    interpretation_module im_instance;
6    image_provider ip_instance;
7    link im_instance to server_instance;
8    link ip_instance to server_instance;
9
10   server_instance.in(ip.out, user_instance.out);
11   im_instance.in(server_instance.out[1]);
12   ip_instance.in(im_instance.out);
13   user_instance.in(server_instance.out[0]);
14 }
```

Listing 2: System description of Chips simplified TeaStore model

4 Chips design of a TeaStore

Model Devise Following CT bases [12], Chips uses block diagrams. In the first time, the system model must be assembled by turning each module of the specifications into a block converting an input signal into another kind of output signal, as illustrated in Figure 1. The choice was made in this paper to keep the model as simple as possible. The use case modeled is one where only one user is conversing with the server. Although it is not realistic, it remains easy both to implement and to improve when the need comes to scale the model up. Some other points are worth noticing, there are some differences between the original modules described in Section 2 and the diagram presented:

- The WebUI has been split into two components to clearly separate its two functions, receiving the data from the user and sending back a response (respectively *User Action Interpreter* and *Web Page Service*).
- The physical interfaces through which data are exchanged are represented (*Server Physical Interface* and *User Computer Interface*).
- A *Request Validator* block is there to decide whether a request for data of the user is correct or not according to its authentication state.

When working on constraints as response time or fault tolerance, it is important to model not only the processed data, but also the context in which it is processed. Hence the differentiation of logical and physical functions of our blocks. In the end, any algorithm will always be executed by a hardware component. This hardware is responsible for the efficiency of the system at a certain task and if we want to keep our model useful, we must take it into account. Figure 2 is complementary to the block diagram described before. The physical dependency graph (associated with Chips *logical* and *physical* keywords and the *link x to y* instruction) allows the model to better determine data transmission timings, calculation speed and memory constraints. When two entities try to communicate while being associated with different physical devices, i.e., transmitting a signal through the physical interaction arrow of the diagram, a compiler could automatically apply a communication protocol set of constraints to keep the model as close to the reality as possible.

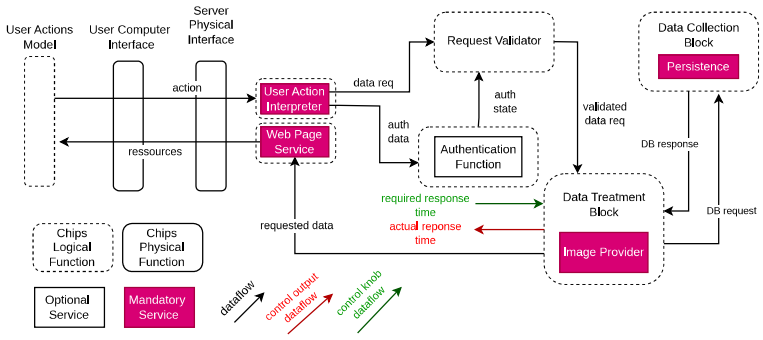

Figure 1: Block diagram of the designed architecture for the TeaStore application and its interaction with a user

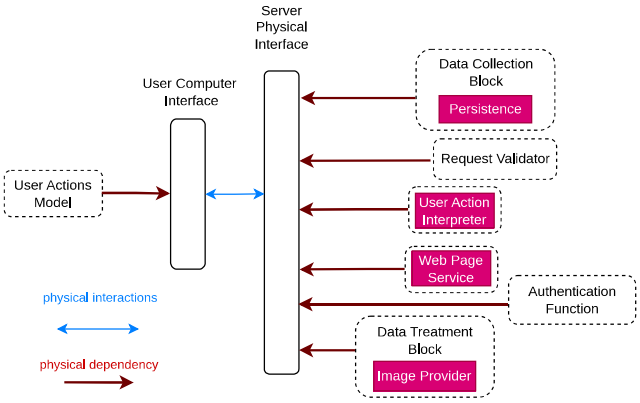

Figure 2: Dependency graph of the designed architecture for the TeaStore application and its interaction with a user

4.1 Modeling hypotheses

To achieve a working model with the aim to parameterize the cache size used by the server, simplifying hypotheses were made:

- The user always waits for an answer from the server.
- The user never disconnects from the server.
- The server provides resources as either `required page` or `authentication page`, modeled by a boolean.
- No recommendation system is used.
- The user always sends actions to the server after receiving resources from it.
- Once the user has provided right authentication data, it is considered by the server as *connected* until the end of the simulation.
- When the user is connected, the server always responds with the `required page`.

- Time delays are negligible, excepts for the ones it takes to fetch images from the cache or from the database.
- The server never shuts down and no packets are lost between it and the user computer.
- The database always contains the requested data, and provides images one at a time.
- The server tries to keep the response time of the image provider around a given constant time value.
- User actions are interpreted as a three fields data structure. `action_request`:

  bool isHeavyRequest
  bool requestPrivatePage
  bool providesRightAuthData

These assumptions allow abstracting away many constraints that would render the system too complex to analyze. If the control were to be applied to another part of the system, such hypotheses would have to be changed to properly separate the study concerns.

4.2 TeaStore adaptable component

The component of interest –the cache of our image provider block– can now be studied without worrying about other sources of disturbances than the one that directly impacts them. To do so, the image provider block includes, on top of the cache, a controller component depicted in Fig. 3.

Controller Design A PID (Proportional Integral Derivative) controller is used to manage the caching part of the TeaStore application. It is a controller that takes into account the error (difference between expected and resulting value) of a system, its integral and its derivative to modify the knob it controls. This kind of controller is used in a vast majority of control scenarios with good results and its simplicity makes it an easy to implement example of controller to work with in our case [3]. The inner data treatment block controller will have the task to convert its input command (the response time required by the server general controller) into a cache size (an integer corresponding to the number of items the cache can retain), so that the overall calculation time of the system remains around the command value. The cache used will apply the Least Recently Used strategy (LRU), and to keep the model lightweight, images will be modeled by an ID number between 1 and the total number of images of the database DB_SIZE. For each request, the image provider will look for the images IDs in the cache. This search is modeled by the $CACHE\_SEARCH\_TIME$ value. For each image not found in the cache, the image provider sends a data request taking DB_REQ_TIME. Therefore:

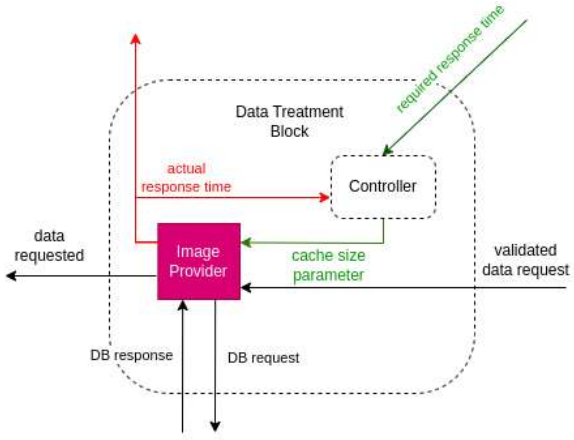

Figure 3: In depth block diagram of the Data Treatment Block

$$actual\_response\_time = CACHE\_SEARCH\_TIME + X \times DB\_REQ\_TIME$$

$$CACHE\_SIZE(t) = P \times error(t) + I \times \left(\int_0^t error(t) \times dt \right) + D \times \frac{d\ error(t)}{dt}$$

where:

- X is the number of images not found in the cache depending on the previous requests and the actual size of the cache,
- P, I and D are constant real values chosen either empirically, arbitrarily by experience of a CT expert, or analytically by solving the whole system equation,
- $error(t) = response\_time\_required - actual\_response\_time$.

If the cache is too small, X is likely to be high, while if $X \geq DB\_SIZE$, the response time will always be $CACHE\_SEARCH\_TIME$ as the cache can contain the whole database. To match the behavior of a user who sometimes requires the same images, and sometimes completely different images, the images requested will be randomly generated in $[1, DB\_SIZE]$. That representation could also be upgraded by changing the probability distribution of the chosen numbers to simulate the popularity of different articles.

4.3 Aggregate programming benefits

If the running system has multi threading capabilities for parallel computing, one could imagine enhancing the previously described version of the controller (see Figure 3) with more instances of image provider. In such case, the control logic can remain the same, but the way the command signal is transmitted to the under-control component must be profiled. To be efficient, such architecture requires to always send the same requests to the same image provider components so the different caches don't store the same information twice. The data has to be split according to rules depending on the components realizing operations with that same data. This concept can be expressed in a quite natural way with Aggregate Programming (AP for short) base block operators [6]. Splitting the data correctly can be done in a two steps algorithm, a first "collecting" operation (the C AP primitive) to count the number of available image providers, then a "spreading" operation (the G primitive) to assign the correctly each image ID to request to the right image provider instance. That work would be accomplished by an additional component interfacing the raw input signal and the profiled versions of it for the image providers (*Request profiling function* in Figure 4). A simpler component would be used to gather the images requested as it would only collect the data from all the image providers answering and compile them into a single signal (*Data aggregating function*).

Thank to these operators, the same encoding of the Data Treatment Block can be used, whatever the number of instances of image provider. This facility of AP to abstract away complex communication and coordination protocol layers is what Chips is seeking for in term of expressivity of its models. It makes easier for a described system to turn the architectural configuration of the system into parameters that the model's controllers can tune, hence allowing even more adaptability. This multi-provider instance model is provided here as a motivation for Chips further development, but could not be implemented as more reflection is needed to introduce AP syntactic constructions without breaking the rigor the synchronous paradigm offers. This point will be expanded in this document "Ongoing Work" Section (Section 8).

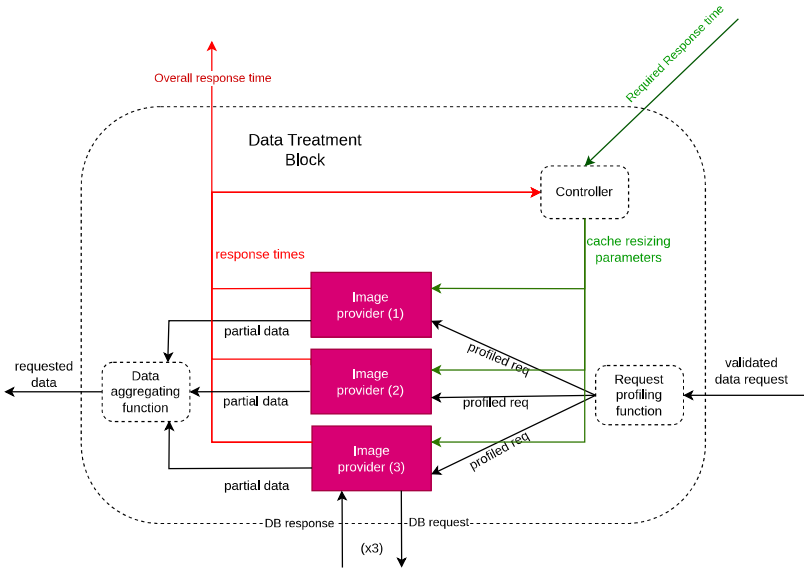

Figure 4: Data Treatment Block handling several Image Provider instances

5 Experimental setup

Test and validation of the system The Chips model for the TeaStore application was manually translated into a BIP set of components according to transformation rules described in Subsection 5.1[1]. The presented experiment was conducted on the BIP model[2] with a machine using a 13th Gen Intel® Core™ i713850HX processor and a RAM of size 32 Gigabytes. The BIP model for the TeaStore application takes about 2 to 3 minutes to compile into an executable code for simulation. And such simulation takes a few seconds to run with the parameter described in Subsection 5.2. Each component of the block diagram implemented is modeled by a single automaton, having at least as many transitions as the block's inputs and outputs.

5.1 Chips to BIP transformation

Most of the functional blocks of the Chips model described by our model (Figure 1) could be turned into an associated BIP automaton in the following manner (also shown by Figure 6):

- Each inner variable, input parameter and output parameter of the Chips function is modeled by a variable of the same name in the BIP automaton.
- The BIP automaton associated to a Chips function is a circular automaton with as many states as the number of inputs plus the number of outputs.
- Each state representing an input parameter has a transition leading to the state representing the next input parameter of the function.

[1]The source code of the Chips Model and its BIP translation is currently available at `https://github.com/NwaitDev/Chips_Public`

[2]The BIP version used is available at `https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/bip/compiler.git`

- Each state representing an output parameter has a transition leading to the state representing the next output parameter of the function.
- The last input parameter state has a transition leading to the state representing the first output parameter.
- The last output parameter state has a transition leading to the state representing the first input parameter.
- Each transition is labelled by the name of the parameter represented by the state it leads to (the parameter is exported by the BIP automaton so the data can be read/written from outside the automaton)
- The *Then* section is translated to BIP and associated to the transition leading to the state representing the last input.

No final state is required as the system isn't supposed to shut down. The initial state is then set to the state representing the last output parameter for every component except for an arbitrarily chosen one. That remaining component will start at the state representing the last input parameter so the system isn't in deadlock. On the initial transition leading the component to its first state, the Chips *init* section is executed. The only component that did not follow this transformation process is the Data Treatment Block. In Listing 1 line 26, the Chips program doesn't model the interaction with a database component. The BIP implementation that was used for the experiment comprises such interaction, but no Chips primitive can currently express it. Therefore an additional state was added. Indeed, the current semantic of Chips only allows to transmit information from a component to another when the *Then* section is fully executed (Figure 5). Sending requests to another component and waiting for its response is currently impossible (or at least, a lot more refactoring has to be done). Designing language constructs to specify inner-*Then* synchronized operation is one of the current concerns of this research and will be more detailed in Section 8.

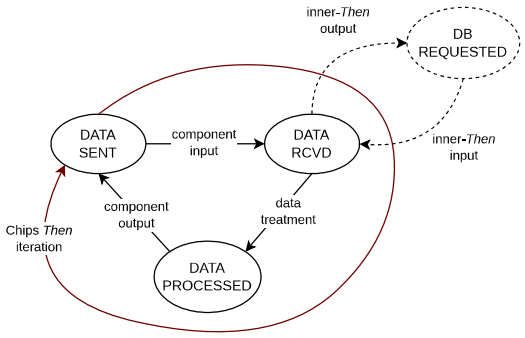

Figure 5: BIP simplified automaton for the image provider algorithm (dashed lines represent elements with no Chips equivalent)

5.2 Parameters of the model

To run this experiments, parameters of the model were arbitrarily chosen. Even if not particularly based on field values or any website response time benchmarks, variables stay relatively coherent in view of assumptions made on the authors' base knowledge on most websites behaviors:

- The cache search takes less time than the image fetch from the database.
- The system aims at reducing the time it takes to provide resources.
- The number of images to provide is greater when the user is connected (to provide profile pictures for instance). The presented experiment will not make use of this parameter as it seems to be useful for a more detailed model only.

```
logical img_provider_controller(
        float requested_time, float resulting_time)
 init {
    float derivative = 0;
    float integral = 0;
    float error = 0;
    float dt = 0;
    int cache_size = 5;
    int minimum_cache_size = 20;
    int maximum_cache_size = 100;
} then {
    dt = resulting_time;
    error = expected - received;
    integral += error * dt;
    derivative = error / dt;
    cache_size = int(pid(error, integral, derivative));
    cache_size = min(maximum_cache_size, cache_size);
    cache_size = max(minimum_cache_size, cache_size);
} -> (cache_size)
```

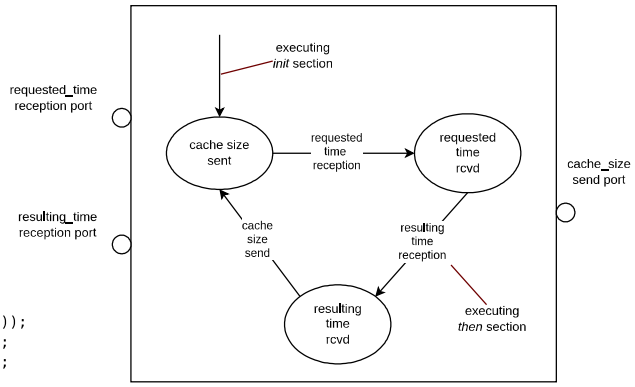

Figure 6: A Chips implementation of a PID controller and its BIP model equivalent

- The cache has a maximum size (no server has an infinite memory) and a minimum cache size. The minimum cache size was arbitrarily set to 20 as setting a cache size "too low" would give an irrelevant hit rate and a website designer would probably better not use any cache than using an almost useless one.

Hence the following parameters:

```
float REQUIRED_RESPONSE_TIME = 4.0;
float CACHE_SEARCH_TIME = 0.3;
float DB_REQ_TIME = 2.0;
float CACHE_MAX_SIZE = 1000;
float CACHE_MIN_SIZE = 20;
int DB_SIZE = 40;
```

5.3 User activity scenario design

In the follow-up results, the user scenario presented is separated in two phases:

- A phase of "wandering", where the user model component only sends a request for two images at the same time. It repeats this operation 300 times.
- A phase of "intensive browsing", where the components send a request for six images and repeat this process until the end of the simulation.

Even if this behavior looks more like integration testing than an actual user behavior, it serves well the purpose of this paper as a proof of concept for Chips model design workflow.

6 Simulation results

Figure 7 shows the behavior of the previously described system model. These curves were obtained using the following parameters for our PID controller: $P = -1, I = -0.01, D = 0$. These PID parameters were empirically tuned until the plots could exhibit satisfying results. Other design techniques could be used as shown in [3].

During the first 800 seconds of the simulation, the user requests are answered by the server under 3 seconds on average, which is coherent regarding the number of images requested by the user during their

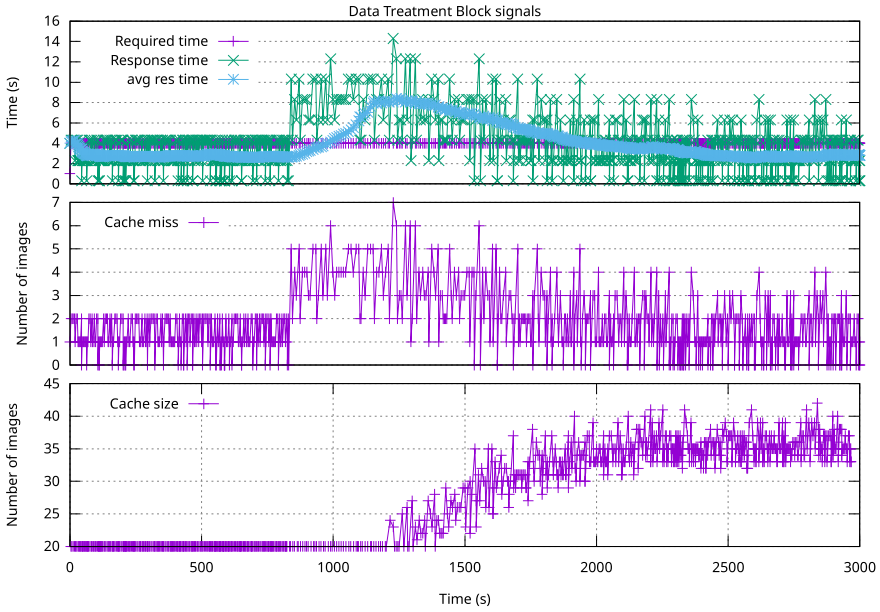

Figure 7: Real time adaptation of the cache size according to the behavior of the client model

"wandering phase". The response time can only be greater than the required response time (4 seconds) when no image is found in the cache (2 cache misses take $2 \times 2.0 + 0.3$ seconds). As half the database can be stored in the cache when it is at its minimum size (20 images), it is likely that images requested by the user can be found, hence the average response time of 2.8 seconds.

When the user disturbs the system by asking for more images for each request ("intensive browsing phase") after the 300 requests (t = 834), the response time increases and the controller adapts the cache size to keep the response time around the command. The jittering phenomenon of the cache size around 37 is due to the fact that our system is discrete. It cannot set the response time exactly to the command so it alternates between a response time a little shorter and a little longer than required.

Let us note that by tuning the PID coefficients, it is possible to change the behavior of the system to reach the optimal cache size quicker, at the risk of temporarily setting the cache size way higher than required. It is 40 in this setup since this is the size of the database and having a bigger cache wouldn't give better results.

7 Conclusion

This paper described a model of adaptive cloud architecture for a variation of the TeaStore web application that has been implemented. The obtained results showed that control theory can effectively be applied to such computer system to improve its performances. Its design followed a full workflow:

- the rigorous analysis of web app goal (keeping the server response time low),
- the distinction of the different means of actions to tune the system in order to achieve these goals (modifying the size of the cache at runtime),
- the design of a control theory block diagram of the system directly translatable to a Chips model (Figure 1),

- the design of a controller within the Chips model for a chosen component (Figure 3),
- the systematic transformation of the Chips model to a BIP one,
- the validation of the system design by simulating the execution of the BIP model and fine-tuning its parameters.

During the development of this sample project, simplifying decisions were made at different levels. The first ones were about the description of the model where the recommendation block was removed and only one user was modeled. Then, the real-time optimization of the cache size was chosen for this study, thus orienting the choice of the information held by the signals exchanged between the user and the server. Finally, the kind of controller used was one of the most classical that exists to keep the demonstration as generic and easy to repeat as possible. According to the designer's will and necessities, other choices can be made to better suit a different project.

8 Ongoing Work

As Chips is still in its early days, most of its development stack is yet to create. Many features are envisioned to make this project applicable. These last paragraphs detail ongoing work on the language.

A complete version of the compiler Currently, the language has a syntax that can be parsed, but not interpreted. No compiler exist for the language, and the hardware description file structure is not determined either. These elements are the current priorities of the Chips project and shall soon be developed. Our intention is to apply ATL [13] transformations to turn a Chips meta-model into the BIP meta-model, thereof, to generate a Chips-to-BIP compiler. Until then, the experimentation presented here could only be realized after a hand-written compilation.

New language constructs Few models were built thank to Chips, and as the language will be applied to more case studies, new needs are expected to emerge in terms of the Chips expressivity. Some interesting features have already appeared though.

- **Building AP primitives**: When Chips includes more complex data structures to represent many components under the same identifier, it will become relevant to provide operators that specify how to spread/gather data among many components. They shall then be translated into BIP components and connectors implementing protocols to simulate AP operations [16]. In AP, operators require to specify *accumulation functions* that are meant to be applied along the spanning tree of communication devices form. The BIP connectors simulating this tree structure for an automata network are currently under development. Their linked list version is currently functional, but it couldn't be added to our model to exhibit significant results in time for the edition of this document. Once the BIP implementation of these AP n-ary connectors is robust enough, there will only remain to find a Chips syntactic construction to make AP completely part of the language core version.
- **Inner-*Then* data synchronization**: As mentioned in the last paragraph of Section 5.1, Chips lacks a way to signify a data is to be read or written during the update of the variables of a component. A possibility would be to introduce a subtype for the inner variables of a component to turn them into some sort of public attributes in the style of object oriented programming. Such decision isn't made yet and has to be confronted with the formalism Chips tries to adopt so the language doesn't become too permissive and prone to errors.

- **Hierarchical control features**: With its building and plugging blocks design, Chips also permits the description of hierarchically organized control. Adding layers of control for a complex system could allow each sub-system to work at its own pace in a correct way while together reaching a larger goal [14]. This layering could be achieved by adding annotations to our Chips blocks for instance. It would explicitly show the encapsulation of signals within a control layer.
- **Virtual clock system**: Finally, Chips models relevance should as well be ensured at the level of the timing constraints of the physical devices they represent. Clocks of a multiple devices system should be handled automatically thank to the use of the hardware description files. The BIP clock component would only serve as a virtual helper for simulation and model checking. It shall be automatically removed at the lowest level compilation of the programs when implementing models on real systems.

References

[1] Antonio Filieri et al. (2015): *Software Engineering Meets Control Theory*. In Paola Inverardi & Bradley R. Schmerl, editors: *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, IEEE Computer Society, pp. 71–82, doi:10.1109/SEAMS.2015.12. Available at https://doi.org/10.1109/SEAMS.2015.12.

[2] Jóakim von Kistowski et al. (2018): *TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research*. In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 223–236, doi:10.1109/MASCOTS.2018.00030.

[3] Karl J. Astrom & T. Hagglund (1995): *Pid Controllers*, 2e édition edition. ISA.

[4] Rim El Ballouli, Saddek Bensalem, Marius Bozga & Joseph Sifakis (2018): *DR-BIP - Programming Dynamic Reconfigurable Systems*. Technical Report TR-2018-3, Verimag Research Report.

[5] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen & Joseph Sifakis (2011): *Rigorous Component-Based System Design Using the BIP Framework*. *IEEE Softw.* 28(3), pp. 41–48, doi:10.1109/MS.2011.27. Available at https://doi.org/10.1109/MS.2011.27.

[6] Jacob Beal & Mirko Viroli (2014): *Building Blocks for Aggregate Programming of Self-Organising Applications*. In: *Eighth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, IEEE Computer Society, pp. 8–13, doi:10.1109/SASOW.2014.6. Available at https://doi.org/10.1109/SASOW.2014.6.

[7] Simon Bliudze, Giuseppe De Palma, Saverio Giallorenzo, Ivan Lanese, Gianluigi Zavattaro & Brice Arléon Zemtsop Ndadji1 (2025): *Adaptable TeaStore*. Available at https://arxiv.org/pdf/2412.16060.

[8] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf & Joseph Sifakis (2010): *From high-level component-based models to distributed implementations*. In: *In Proc. Tenth ACM Int. Conf. on Embedded Software*, EMSOFT '10, Association for Computing Machinery, New York, NY, USA, pp. 209–218, doi:10.1145/1879021.1879049. Available at https://doi.org/10.1145/1879021.1879049.

[9] Antonio Cansado, Carlos Canal, Gwen Salaün & Javier Cubo (2010): *A Formal Framework for Structural Reconfiguration of Components under Behavioural Adaptation*. *Electronic Notes in Theoretical Computer Science* 263, pp. 95–110, doi:https://doi.org/10.1016/j.entcs.2010.05.006. Available at https://www.sciencedirect.com/science/article/pii/S1571066110000460. Proceedings of the 6th International Workshop on Formal Aspects of Component Software (FACS 2009).

[10] P. Caspi, D. Pilaud, N. Halbwachs & J. A. Plaice (1987): *LUSTRE: a declarative language for real-time programming*. In: *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Pro-*

gramming Languages, POPL '87, Association for Computing Machinery, New York, NY, USA, p. 178–188, doi:10.1145/41625.41641. Available at https://doi.org/10.1145/41625.41641.

[11] Gwenaël Delaval, Eric Rutten & Hervé Marchand (2013): *Integrating Discrete Controller Synthesis into a Reactive Programming Language Compiler*. *Discrete Event Dynamic Systems* 23, pp. 385–418, doi:10.1007/s10626-013-0163-5.

[12] Joseph L Hellerstein, Yixin Diao, Sujay Parekh & Dawn M Tilbury (2004): *Feedback control of computing systems*. John Wiley & Sons.

[13] Frédéric Jouault, Freddy Allilaire, Jean Bézivin & Ivan Kurtev (2008): *ATL: A model transformation tool*. *Science of Computer Programming* 72(1), pp. 31–39, doi:10.1016/j.scico.2007.08.002. Available at https://www.sciencedirect.com/science/article/pii/S0167642308000439.

[14] Riccardo Scattolini (2009): *Architectures for distributed and hierarchical Model Predictive Control – A review*. *Journal of Process Control* 19(5), pp. 723–731, doi:https://doi.org/10.1016/j.jprocont.2009.02.003. Available at https://www.sciencedirect.com/science/article/pii/S0959152409000353.

[15] Loïc Sylvestre, Jocelyn Sérot & Emmanuel Chailloux (2024): *Programming parallelism on FPGAs with Eclat*. In Massimo Torquati, editor: *Proceedings of 17th International Symposia on High-Level Parallel Programming and Applications HLPP 2024*, Proceedings of 17th INTERNATIONAL SYMPOSIUM ON HIGH-LEVEL PARALLEL PROGRAMMING AND APPLICATIONS HLPP 2024, Massimo Torquati and Marco Danelutto, Pisa, Italy, pp. pp. 69–88. Available at https://hal.science/hal-04772531.

[16] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei & Danilo Pianini (2019): *From distributed coordination to field calculus and aggregate computing*. *Journal of Logical and Algebraic Methods in Programming* 109, p. 100486, doi:10.1016/j.jlamp.2019.100486. Available at https://www.sciencedirect.com/science/article/pii/S235222081930032X.

[17] E. Zuazua & Enrique Fernández Cara (2003): *Control theory: history, mathematical achievements and perspectives*. 26, pp. 79–140. Available at https://api.semanticscholar.org/CorpusID:16939188.